(19) Canadian Intellectual Property Office
An Agency of Industry Canada

Office de la Propri,t, Intellectuelle du Canada
Un organisme d'Industrie Canada

(11) CA 2 290 265 (13) A1

(40) 24.05.2001
(43) 24.05.2001

(12)

(21) 2 290 265

(51) Int. Cl.7: H04L 12/56, H04L 12/24

(22) 24.11.1999

(71)
HASHEMI, MASSOUD,
815 - 35 Trailwood Drive, MISSISSAUGA, O1 (CA).
LEON-GARCIA, ALBERTO,
8 Longspur Road, TORONTO, O1 (CA).

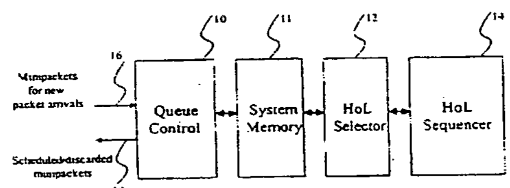
(72)
HASHEMI, MASSOUD (CA).
LEON-GARCIA, ALBERTO (CA).

(74)
FASKEN MARTINEAU DUMOULIN LLP

(54) ORDONNANCEUR PROGRAMMABLE HAUTE VITESSE DE PAQUETS DE DONNEES ET GESTIONNAIRE DE MEMOIRE TAMPON

(54) HIGH-SPEED PROGRAMMABLE PACKET SCHEDULER AND BUFFER MANAGER

(57)
A method and apparatus for managing the buffering and for scheduling the transfer of packets of data arriving on one or a plurality of input ports and destined for one or a plurality of output ports of a packet switch or router or a subsystem thereof is disclosed. Each arriving packet is assigned an index that specifies both a unique destination output port for the packet and membership in a subclass, such as a priority class, a connection or flow, or a class of packets that is to receive a certain type of transmission or buffering service. For each arriving packet, a minipacket is created that contains the index of the packet, a unique identifier such as a storage location, and optionally information relevant to the scheduling and buffering of the packet. Each index is assigned a unique queue in which minipackets with the given index are placed in order of arrival. A method and apparatus for a scheduler and buffer manager is disclosed that: maintains a large number of said queues; identifies the head-of line minipackets from said queues to a head-of line sequencer ; selects the order in which minipackets are output according to a scheduling algorithm. Packets are transmitted from storage to output ports according to the sequence of minipackets that exit the scheduler system. The disclosed scheduler is flexible and can be programmed to implement various scheduling algorithms, including weighted fair queuing, priority queuing, hierarchical fair queuing and all variations that involve sorting according to a tag value. The disclosed scheduler is scalable in that it can provide individual queues for a very large number of port/subclass combinations. The disclosed scheduler can be implemented in integrated circuits to operate at very high speeds.



OPIC
Office de la propriété intellectuelle du Canada

CIPO
Canadian Intellectual Property Office

(12) (19) (CA) Demande-Application

(21) (A1) 2,290,265

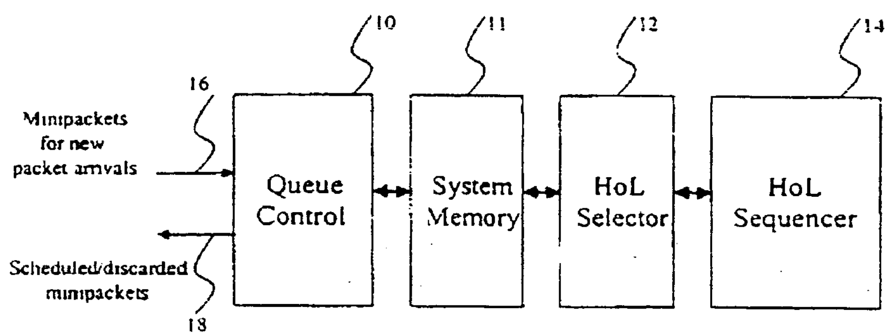
(22) 1999/11/24
(43) 2001/05/24

(72) LEON-GARCIA, ALBERTO, CA
(72) HASHEMI, MASSOUD, CA
(71) LEON-GARCIA, ALBERTO, CA
(71) HASHEMI, MASSOUD, CA

(51) Int.Cl.[7] H04L 12/56, H04L 12/24

(54) ORDONNANCEUR PROGRAMMABLE HAUTE VITESSE DE PAQUETS DE DONNEES ET GESTIONNAIRE DE MEMOIRE TAMPON

(54) HIGH-SPEED PROGRAMMABLE PACKET SCHEDULER AND BUFFER MANAGER



(57) A method and apparatus for managing the buffering and for scheduling the transfer of packets of data arriving on one or a plurality of input ports and destined for one or a plurality of output ports of a packet switch or router or a subsystem thereof is disclosed. Each arriving packet is assigned an index that specifies both a unique destination output port for the packet and membership in a subclass, such as a priority class, a connection or flow, or a class of packets that is to receive a certain type of transmission or buffering service. For each arriving packet, a minipacket is created that contains the index of the packet, a unique identifier such as a storage location, and optionally information relevant to the scheduling and buffering of the packet. Each index is assigned a unique queue in which minipackets with the given index are placed in order of arrival. A method and apparatus for a scheduler and buffer manager is disclosed that: maintains a large number of said queues; identifies the head-of line minipackets from said queues to a head-of line sequencer ; selects the order in which minipackets are output according to a scheduling algorithm. Packets are transmitted from storage to output ports according to the sequence of minipackets that exit the scheduler system. The disclosed scheduler is flexible and can be programmed to implement various scheduling algorithms, including weighted fair queuing, priority queuing, hierarchical fair queuing and all variations that involve sorting according to a tag value. The disclosed scheduler is scalable in that it can provide individual queues for a very large number of port/subclass combinations. The disclosed scheduler can be implemented in integrated circuits to operate at very high speeds.

Industrie Canada Industry Canada

ABSTRACT

A method and apparatus for managing the buffering and for scheduling the transfer of packets of data arriving on one or a plurality of input ports and destined for one or a plurality of output ports of a packet switch or router or a subsystem thereof is disclosed. Each arriving packet is assigned an index that specifies both a unique destination output port for the packet and membership in a subclass, such as a priority class, a connection or flow, or a class of packets that is to receive a certain type of transmission or buffering service. For each arriving packet, a minipacket is created that contains the index of the packet, a unique identifier such as a storage location, and optionally information relevant to the scheduling and buffering of the packet. Each index is assigned a unique queue in which minipackets with the given index are placed in order of arrival. A method and apparatus for a scheduler and buffer manager is disclosed that: maintains a large number of said queues; identifies the head-of-line minipackets from said queues to a head-of-line sequencer ; selects the order in which minipackets are output according to a scheduling algorithm. Packets are transmitted from storage to output ports according to the sequence of minipackets that exit the scheduler system. The disclosed scheduler is flexible and can be programmed to implement various scheduling algorithms, including weighted fair queuing, priority queuing, hierarchical fair queuing and all variations that involve sorting according to a tag value. The disclosed scheduler is scalable in that it can provide individual queues for a very large number of port/subclass combinations. The disclosed scheduler can be implemented in integrated circuits to operate at very high speeds.

A HIGH-SPEED, PROGRAMMABLE PACKET SCHEDULER AND BUFFER MANAGER

The present invention in general relates to high-speed packet switches and routers for use in telecommunication and computer networks, more specifically, the present invention relates to queuing and scheduling in switches and routers that provide differential transfer service to multiple subclasses of packets destined to a given output port. The present invention can be used in switches and routers to provide quality of service across packet networks.

BACKGROUND OF THE INVENTION

A data packet is a discrete quantity of information that is normally sent serially between devices connected via a network. Packet switches and routers are used to direct these packets along the various branches of a network to its destination, using information contained in the packet header.

In switches and routers data packets arriving from input ports are buffered and after routing and classification, that is done by processing the packet headers, the packets are queued and scheduled for transmission to individual output ports. Queuing is required in switches and routers because more than one packet may arrive from different input ports for the same output port. The packets have to be queued and sent from the output port one at a time. Packets destined for an output port may belong to different applications such as data, voice, video and so on.

A desirable feature of packet networks is the capability to buffer and transfer packets differentially in a manner that may depend on the type of application. Different applications can require different qualities of service (QoS) in the transfer of packets. The quality of a service can be specified by the fraction of packets that are lost, by the total packet delay, and by the variations in the total packet delay (jitter) that packets experience in traversing switches and routers across the network.

To provide differential buffering and transfer service in a switch or router, packets that are destined to a given output port are classified according to priority level, connection or flow, or membership in some subclass of packets that require a certain type of service. The packets of different classes or priorities may then be put in different queues. A scheduler is used to select the packets from these queues for transfer to the output port according to some scheduling algorithm. The order in which the queued packets are selected for transmission to an output port affects the loss, delay, and jitter that a packet incurs in a switch or router.

The relative preference that different packet classes are given in access to buffering also affects the loss probability. Packets that are given preferential access will in general experience lower loss than other packets. Buffer management algorithms are used to implement differential access to buffering as well as to discard packets under certain conditions.

Queuing and scheduling and buffer management are key parts of any switch or router that is designed to provide differential packet buffering and transfer in a packet network. Different network architectures use different approaches to providing differential service and so impose different requirements on the queuing and scheduling system.

Traditional routers in Internet Protocol networks provide only best effort packet transfer, where there is no guarantee that a packet will be delivered to its destination, and where no guarantees are made regarding measures such as packet loss, packet delay, or delay jitter. Best effort packet transfer does not provide differential treatment to different classes of packets, and so first-in first-out (FIFO) queuing and packet transfer is adequate in such routers. The dramatic growth in World Wide Web traffic has made the Internet the preferred communications network for deploying new applications. The infrastructure of the Internet must therefore be modified to support applications such as telephony, real-time audio and video, streaming audio and video, and interactive

applications such as games and auctions. However different types of applications have different quality of service requirements, and so the infrastructure must be capable of providing differential packet transfer service to the packet streams of different applications.

Differential buffer access can be provided in a FIFO queuing system to provide different levels of packet loss to different packet types . For example, if there are two types of packet traffic, both classes are allowed access if the number of packets in queue is below a first threshold value. When the number of packets exceeds the first threshold value, only the first packet type is allowed access and second- packet type arrivals are discarded. Once the number of packets exceeds a second higher threshold, all packet arrivals are discarded.

Buffer management algorithms, such as Random Early Detection (RED) [Floyd 1993], are designed to throttle the rate at which end systems send packets into a network. In the case of RED, if the average number of packets in queue is greater than a first threshold, then packets are discarded according to a probability that depends on the margin, which exceeds the threshold. The discarded packets are intended to signal to an endsystem mechanism such as TCP to reduce the rate at which packets are sent into the network before congestion sets in. Once the number of packets exceeds a second threshold all arriving packets are discarded.

Multiple instances of RED can be used to provide different packet loss performance for different types of packets. Each packet type has a corresponding pair of threshold values and associated discard probabilities. Each packet arrival is then discarded according to the current average number of packets in queue and the –type-specific thresholds and discard probabilities.

Policing mechanisms are used to enforce the manner in which packets from a given class are allowed to arrive at a system. For example, the peak and average arrival rate in bytes per second may be enforced by a policing mechanism. The maximum burst size of a

packet arrival in bytes may also be enforced. When packets arrive that violate said arrival pattern, the policing mechanism may discard the packets. Alternatively, the policing mechanism may mark a packet as being non-conforming and indicating that it is to undergo certain treatment in the case of congestion.

The simplest approach to providing differential packet transfer service and hence differential delay performance is to use static priority classes where packets are classified into a small number of classes [Keshav, pg. 223]. Packets destined to a given output port are placed in a queue dedicated to its given priority class. Each time a packet transmission is completed in an output port, the next packet to be transmitted is selected from the head of the line of the highest-priority non-empty queue. As a result packets from higher-priority classes experience lower delay than packets from lower-priority classes. However, large arrival rates of higher-priority packets can result in an insufficient number of transmission opportunities for the lower-priority queues, which can eventually fill up with packets and overflow, resulting in packet loss and excessive delay for the lower priority packets.

An important concern in providing differential transfer service is that different subclasses of packets receive a fair share of packet transmission opportunities. Round robin scheduling is a mechanism for providing an equitable share of transmission opportunities [Peterson, pg. 403]. In round robin scheduling each packet class has its own queue, and the scheduler selects the head-of-line packet from each queue in round robin fashion, so that after each round each queue has had one transmission opportunity. If packets from each subclass have the same average packet length, then over the long run each queue will transmit the same volume of information, measured in bits. However, round robin scheduling can be unfair. If packets from different subclasses have different average lengths then the volume of information transmitted over the long run by different queues will not be equal.

More sophisticated scheduling schemes such as fair queuing can ensure fair sharing of long-run transmission volumes through mechanisms that takes packet length into account

[Keshav, pg. 239]. Weighted fair queuing can guarantee pre-assigned but not necessarily equal levels of long-run transmission volumes to different subclasses of packets. Weighted fair queuing scheduling has been shown to provide guaranteed bounds on packet delay transfer across a network under certain conditions [Zhang, 1995]. Fair queuing, weighted fair queuing, self-clocked fair queuing and other scheduling algorithms have been shown to be implementable using a dynamic sorting mechanism where each packet is assigned a dynamically-computed tag which is then sorted to find its placement in a queue [Zhang, 1995].

Different network architectures use different approaches to providing quality of service and so impose different requirements on the packet scheduler. ATM networks provide quality of service guarantees to every virtual connection of cells (fixed-length packets) that is established across the network. This can require that the cells of each connection be queued separately, called per-VC queuing, and that the scheduler be aware of every connection in its corresponding output port. Integrated Services IP routers can provide packet transfer delay guarantees but they must place packets for each separate packet flow in a separate queue and the scheduler must be aware of the connections in an output port [Peterson, pg. 464]. The requirements that ATM and Integrated Services IP handle individual packet flows imply that schedulers must be able to handle a very large number of queues for the different flows. To provide delay guarantees, weighted fair queuing scheduling mechanisms are required in ATM switches and Integrated Services IP routers.

Differentiated services IP routers provide differential transfer to packets according to their membership in specified classes. A field in the IP header is used to indicate that a packet is to experience a certain type of per-hop behavior (PHB). A PHB is the externally observable behavior of a packet as it traverses a router. Various types of scheduling algorithms can be used to produce the standardized behaviors. Six bits have been allocated to identify different PHBs, so a differentiated service router may be required to handle up to 64 separate classes. PHBs address not only delay behavior but also packet loss performance. In particular, different PHBs can be defined to have different levels of packet drop precedence.

In differentiated services IP, a customer and a service provider may establish a traffic agreement that specifies the temporal properties of the traffic stream that the customer is allowed to transmit into the network. Two service providers may also establish such agreements to exchange traffic. Users and service providers may apply shaping mechanisms to their traffic to ensure conformance to the agreement.

Some routers are required to allocate transmission opportunities on a hierarchical basis. For example, a router may be required to share the transmission on an output port first according to organization, and within each organization, according to packet flow or application type [Floyd 1995]. Hierarchical packet fair queuing algorithms [Bennett 1996] can provide this type of allocation. Hierarchical allocation of transmission is an important requirement in the partitioning of packet networks into private virtual networks that have dedicated amounts of transmission allocated to them.

Virtual networks can be created by partitioning the transmission capability of an output port among a number of virtual networks using a mechanism such as Multiprotocol Label Switching [Davie]. Each virtual network may handle its own set of packet flows. A scheduling algorithm can be used to enforce the partitioning of buffering and bandwidth in a router and if necessary to provide the differential treatment to different packet flows within each virtual network. Routers that implement MPLS are intended for operation in the core of the Internet where very high-speed operation will be required. The number of MPLS flows that need to be handled in a large backbone network can be very large.

Connection-oriented networks such as ATM provide quality of service guarantees to every virtual connection of cells (fixed-length packets) that is established across the network. This can require that the cells of each connection be queued separately, called per-VC queuing, and that the scheduler be aware of every connection in its corresponding output port. The requirement that ATM handle individual packet flows implies that schedulers must be able to handle a very large number of queues for the different flows. To provide end-to-end delay guarantees, shaping and scheduling mechanisms need to be

set up in the switches along a connection. For example, U.S. Patent 5,864,540 and [Rexford et al, 1997] uses an integrated shaping and scheduling mechanism to handle fixed-length ATM cell connections with specified rate and burstiness traffic parameters.

In summary, an ideal packet scheduler should be able to: 1. Implement the scheduling algorithm that is appropriate in a given network scenario; 2. Schedule large to very large numbers of packet flows depending on the network scenario; and 3. Be capable of operating at very high speed.

Schedulers that can operate at high speed and for a large number of queues are difficult to implement. Many existing switches and routers implement queue scheduling in software. Software-based scheduling is programmable, but does not scale to very high speed [Kumar], [Newman], [Keshav 1998]. Existing hardware-based schedulers can achieve high speeds but have limited flexibility. Most switches that provide scheduling in hardware provide only a small number of priority queues (typically 2 to 8) and use an arbiter that services the priority queues using round-robin or weighted round-robin methods [Hluchyj, "Methods for prioritizing, selectively discarding, and multiplexing differing traffic types fast packets," US Patent 5231633], [KaiEng, "Controller for input-queued packet switch," US Patent 5255265].

Many scheduling algorithms are implementable using a dynamic sorting mechanism where packets are assigned numerical tags that are sorted to find the packet's placement in a queue [Zhang, 1995]. Therefore sequencer circuits have been developed to provide hardware implementations of scheduling algorithms.

In the sorting circuit described in the US patent 4,991,134 entitled "Concurrent Sorting Apparatus and Method Using FIFO Stacks," packets enter the queue from the tail of the queue and have to travel to the head of the queue through sorting steps before being able to leave the queue. Also, the sorting is based specifically on the magnitude of data blocks. The sorting circuits described in the US patent 5,504,919 entitled "Sorter Structure Based on Shiftable Content Memory," and US patent 5,313,579 entitled "B-

ISDN Sequencer Chip Device," use a bus structure to sort the data. All sorting circuits which use a bus structure including the above mentioned ones, suffer from a high latency in operation at each data arrival because of the time required for propagation of the result of the comparison at each unit to the rest of the queue. A new cell can enter only after the settlement of the previous cell in the queue. Therefore the arrival rate cannot be high. Also, in the mentioned circuits the scheduler is limited to certain suggested algorithms that are not necessarily appropriate to certain network scenarios.

The apparent complexity of exact sorting implementations has led to designs that use approximate sorting techniques. For example U.S. Patent 5,864,540 uses a calendar queue, which gives approximate sorting, to shape and schedule traffic in an ATM network to provide conformance with a traffic contract.

A method of achieving low latency exact sorting is presented in U.S. patents #5,274,642, #5,406,556, and #5,440,553, "Output-Buffered Packet Switch with a Flexible Buffer Management Scheme." The method involves inserting new packets from the front of the queue since the only candidates for departure are the packet that is currently at the head of the queue and the new arrivals. A series of research publications has developed packet schedulers using the exact-sorting insertion from the front method. [Hashemi 1997a] describes a sequencer that can implement the scheduling of packets arriving from a plurality of input ports and destined for a single output port. The sequencer can implement any scheduling algorithm that involves the comparison of a tag. [Hashemi 1997b] shows that the sequencer circuit can be programmed to implement a variety of scheduling algorithms, including priority queues, windowed-priority queues, fair queuing, pacing mechanisms, and hybrid combinations of scheduling algorithms. [Hashemi 1997c] describes a "single queue switch" sequencer that can schedule packets arriving from a plurality of input ports and destined to a plurality of output ports. Said sequencer can be used as a centralized scheduler in a memory-based packet switch [Leon-Garcia and Hashemi, "The Single Queue Switch", Canadian Application No. 2,227,655]. Hashemi [1997d] describes how the single-queue switch can be used to

schedule variable length packets. [Zhang 1999] shows that very high speeds can be achieved by integrated circuit implementations of the single queue switch.

One feature of the insert-from-the-front sequencer is the serial comparison of numerical or logical relationships between binary numbers that are attached to minicells or minipackets that enter the sequencer.

Conventional implementations of scheduling algorithms using sequencer circuits require that every arriving packet be represented by a minicell in the sequencer. The sequencer must therefore be able to immediately accommodate all packets that arrive to the system and this requirement places a limit on the number queues (indices) that can be handled. In general the total number of packets in the system is much larger than the number of queues.

A major drawback of all sequencer circuits is scalability in terms of the number of flows that can be scheduled individually. The total number of packets that can be buffered is limited by the number of comparison cells that can be built into an integrated sequencer circuit [Zhang 1999]. The number of packet classes is typically much smaller than the total number of packets in the system. For this reason sequencer circuits have not been used in packet schedulers.

The integrated system of U.S. Patent #5,864,540 and [Rexford et al, 1997] handles the shaping and scheduling of ATM cells directly. This system uses a structure involving the tandem combination of a bank of per-connection FIFO queues followed by several calendar queues that implement approximate sorting. In order to provide different transmission rates to different connections, the integrated system place a number of cells in the sorting unit in proportion to the required bandwidth.

SUMMARY OF THE INVENTION

An insert-from-the-front sequencer is the serial comparison of numerical or logical relationships between binary numbers that are attached to minicells or minipackets that enter the sequencer. Said sequencer circuits can be made programmable by placing a superset of comparison logic that can be selected according to the type of scheduling that is desired. The packet scheduler in the present invention provides programmability and high speed by incorporating said type of sequencer circuits and enhancements thereof.

In the present invention a large number of queues can be implemented due to a new approach in which packets are queued in linked-list queues and scheduled by a single-queue sequencer. The method of combining the queuing and scheduling parts makes it possible to schedule a large number of queues using a smaller sequencer.

The present invention provides a method and apparatus for managing the buffering and scheduling the transfer of packets of data arriving on one or a plurality of input ports and destined for one or a plurality of output ports of a switch or router or a subsystem thereof.

Each arriving packet has an index that specifies both a unique destination output port for the packet and membership in a subclass. For each arriving packet, a minipacket is created that contains the index of the packet, a unique identifier such as the packet's storage location, and information relevant to the scheduling of the packet. Minipackets are input into a queue control part that stores said minipackets in order of arrival in a unique queue that is assigned to each index. These queues are implemented using a bank of linked-list queues that can be assigned flexibly and arbitrarily to the minipackets in the system. The bank of linked-list queues is scalable in that it can handle a very large number of queues. The queue control part can implement a variety of buffer management and policing mechanisms.

A head-of-line (HoL) selector part identifies the head-of-line minipacket in a queue and at an appropriate time creates a shorter data unit called a minicell that contains a pointer

to the minipacket and the essential scheduling information. At said appropriate time the HoL selector transfers the head-of-line minicell to a head-of-line (HoL) sequencer part. The HoL selector part replaces each minicell that exits the HoL sequencer part with the HoL minicell from the queue with the same index. This ensures that the HoL sequencing part always has one HoL minicell for each non-empty queue.

The HoL sequencer part determines which minicell should be scheduled for transfer out of the system according to a selected scheduling algorithm. A sequencer circuit determines the order in which the HoL minicells should be transferred out of the one or plurality of output ports. A generalized sequencer circuit is disclosed that can implement any scheduling algorithm that can be translated into a logical or numerical relationship between a series of numbers carried in the minicells. A tagging unit that precedes the HoL sequencer circuit can assign such numbers to the minicells according to the desired scheduling algorithm.

Each time a minicell exits the HoL sequencer, the associated minipacket is dequeued and output from the scheduler system. The actual packets in the switch, router, or subsystem thereof are transferred from storage to output ports according to the sequence of minipackets that exit the scheduler system.

Conventional implementations of scheduling algorithms using sequencer circuits require that every arriving packet be represented by a minicell in the sequencer. The sequencer must therefore be able to immediately accommodate all packets that arrive to the system and this requirement places a limit on the number queues (indices) that can be handled. In general the total number of packets in the system is much larger than the number of queues. The present invention requires that only the head-of-line minicell be present in the HoL sequencer. The number of queues (indices) that can be handled by the system is then equal to the number of minicells that can be accommodated in the sequencer.

In a preferred embodiment the packet scheduler of the present invention handles a sequence of data units call minipackets to control the buffering and transfer of packets

outside the scheduler system. Preferably, the packet scheduler implements exact sorting of all packets in the system by arranging them in a bank of FIFO queues in tandem with a sequencer that contains the single head-of-line minipacket from each non-empty FIFO queue. Provisioning of different transmission rates to the packets of different FIFO queues may be accomplished through the sorting of minipackets according to tags. The choice of sequencer and the structure of the tag determine the type of scheduling that is implemented.

BRIEF DESCRIPTION OF THE DRAWINGS

An embodiment of the invention will now be described by way of example only, with reference to the accompanying drawings in which:

Figure 1 **Block diagram of a scheduler system;**

Figure 2 **Queue controller;**

Figure 3 **System Memory;**

Figure 4 **Example Implementation of Packet Scheduler System;**

Figure 5 **HoL Selector ;**

Figure 6 **HoL Sequencer ;**

Figure 7 **Generalized Sequencer Circuit;**

Figure 8 **Subqueues with Sequencer Circuit;**

Figure 9 **Generalized Single-Queue Sequencer Circuit;**

Figure 10 **Example interleaving the logical output queues for 8 ports;**

Figure 11 **HoL Sequencer with Multicast Controller;**

Figure 12 **Example application of Scheduling System in Centralized Switch;**

Figure 13 **Example application of Scheduler system in Multiport Line Card;**

Figure 14 **Example application of scheduler system in Single port line card**; and

Figure 15 is a schematic diagram of a queue and scheduling system in an input line card.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Figure 1 shows the block diagram of a scheduler system, according to the present invention, that is used for scheduling the transfer of data packets arriving on one or a plurality of input ports and destined for one or a plurality of output ports in a switch, router, or subsystem thereof. The scheduler system of the present invention has an input interface **16** through which it receives data units, called minipackets, that represent data packets that have arrived from input ports and that are temporarily placed in a storage location outside the scheduler system. The scheduler system accepts these minipackets, performs a scheduling algorithm, and outputs on output interface **18** minipackets that specify the order in which data packets are to be transmitted on the output ports. The scheduler system may optionally perform buffer management and packet policing functions in which case it may output minipackets from time to time that indicate that the corresponding data packets are to be discarded.

As seen in Figure 1, the system of the present invention can be viewed as a combination of four major parts: a queue control part **10**; a system memory part **11**; a head-of-line

(HoL) selector part 12; and a head-of-line (HoL) sequencer part 14. The objective of the queue control part 10 is to accept minipackets as input, and to place these minipackets in the queue indicated by the queue index in each minipacket. The system memory part 11 is used for the storage of minipackets and as a repository of queue state information. An important aspect of this invention is that all packets of the same queue index are to be transmitted in first-in first-out (FIFO) fashion. As a result the queue control part can place minipackets in their corresponding queue in order of arrival. An important aspect of this invention is that, at any given time, the minipacket that is selected for transmission must belong to the set of minipackets that are at the head of their respective queue. The objective of the HoL selector part 12 is to ensure that every minipacket from the head of line of each non-empty queue is represented in the HoL sequencer part. At the appropriate time the HoL selector creates a shorter data unit called a minicell that contains a pointer to the minipacket and the essential scheduling information. The HoL sequencer part 14 selects the next minipacket to be transmitted from among the set of all HoL minipackets according to a given scheduling algorithm. In order for the HoL sequencer to consider all eligible HoL minipackets, an important requirement of this invention is that a minipacket arriving to an empty system be transferred to the HoL sequencer directly.

We assume that minipackets are created outside the scheduler system, and that they enter the system sequentially as shown in Figure 1. Each minipacket is prepared so that it contains information required by the queuing and scheduling algorithms. Said information can include priority level or class, packet length, time stamp, and other information, some of which could be optional or not implemented in some realizations. Said information can be extracted directly from the header of the data packets or indirectly by processing of the headers by a packet processor or classifier. In particular each minipacket contains a queue index that specifies the output port that the corresponding data packet is destined for, and optionally membership of the data packet in a subclass. The specific format of minipackets is not of particular importance in this invention; however all the minipackets should have the same format and the realization of the system should also conform to the selected format.

Each minipacket represents a data packet, and so each minipacket must also contain a unique identifier that specifies its corresponding data packet. For example the unique identifier may consist of the storage location of the data packet. The present invention is not limited to any particular data packet format. Minipackets may represent different types of data packets such as IP packets, ATM cells, or ethernet frames. The data packets can be fixed-length such as in ATM, or variable-length such as in IP.

The queue control part **10** accepts minipackets on input **16.** Each input minipacket is stored in memory allocated for this purpose. The queue control part **10** is responsible for keeping track of the order of each minipacket in each queue. The queue control part **10** is also responsible for the output of minipackets. When the HoL sequencer part **14** indicates that a certain data packet should be transmitted next, the corresponding minicell is transferred from the HoL sequencer to the queue control part. The minicell contains a pointer that specifies the location of the corresponding minipacket in memory. The queue control part removes the minipacket from memory, performs certain bookkeeping functions associated with the given minipacket, and sends the minipacket from the scheduler system on output **18**. The data packet corresponding to the given minipacket can then be retrieved from storage and transmitted on the corresponding output port(s).

Table I below is a pseudocode representation of the Queue Control

```
Queue Control


If      (there is an arriving minipacket)    then

        Get the minipacket
        Store the minipacket in memory
        (optional) Store the time stamp
```

```
        (optional) Perform Policing
        (optional) Perform metering
        (optional) Perform buffer management

        If      (queue is not represented in the HOL sequencer) then

                Send the minicell directly to the HOL Selector
                Mark the queue as represented in the HOL sequencer

        Else

                Put the minipacket in the FIFO queue specified by the index
                Update the queue length
                Update the queue age

                If      (queue was empty)    then

                        Mark the queue as nonempty

                Endif

        Endif

Endif

If      (there is a winner minicell)  then
                remove the corresponding minipacket from memory
                send the minipacket out of the scheduler system
```

Endif

In the present invention the plurality of queues are not pre-assigned to any particular service, priority, class, flow or any other category of packets. The use of queues and their assignment to different categories of packets is arbitrary and can be decided by previous stages of the switch or router, or by the controller software in the switch or router. However the mechanism of the assignment of queues to different sets of packets which is an important aspect of the present invention, is as follows.

Each queue is identified by an index. A controller outside the scheduler system, such as a switch/router control, or management software, assigns the queues to different categories of packets. The controller also keeps a record of the assignment in its memory, most likely in the form of a lookup table.

The target queue for a packet needs to be determined before corresponding the minipacket can be sent to the scheduler system. The queue can be chosen by a packet processor, a packet classifier, or other means depending on the switch or router architecture. The category to which the packet belongs is determined from the information attached to the packet. Based on this information, and according to the queue assignment information, a queue is selected for the packet and the corresponding index is attached to the minipacket.

Figure 2 shows the queue control part **10** that performs the function of enqueueing and dequeueing of the minipackets into the queues. The queue control part includes a linked-list controller **20** that maintains the storage of minipackets in the queues. Minipackets arriving to the scheduler system of the present invention are stored in a minipacket memory **30** in the system memory part, **11** in Figure 3. The linked list controller **20** in contains a memory space controller **21** that provides pointers to free spaces in the memory that is used to store arriving minipackets. The pointers are returned to the memory space controller whenever the minipackets are sent out to the fabric. A pool of

pointers or a linked list queue of free blocks of memory can be used by the memory space controller to provide pointers to free blocks in the memory. In certain implementations, the linked list controller may optionally use as the unique identifier that arrives with an input minipacket as pointer to a minipacket memory space.

A minipacket must be placed in the appropriate queue after it has been stored in the minipacket memory **30** in Figure 3. To queue the minipacket, the pointer of the storage location of the minipacket is linked to the linked-list queue identified by the queue index attached to the minipacket. Each linked-list is accessed through two pointers called head pointer and tail pointer that are stored in two buffers known as head pointer buffer **31** and tail pointer buffer **33** as shown in Figure 3.The head pointer buffers are in a separate memory. The head pointer buffer of a linked-list queue stores the pointer of the first minipacket of the queue. The tail pointer buffers are in another separate memory. The tail pointer buffer of a linked-list queue stores the pointer of the last minipacket of the queue. The head and tail pointers of a linked-list queue are determined using the queue index. The head pointer buffer is used to read the head-of-line minipacket out of the linked list queue. The tail pointer is used to write a new minipacket into the linked list queue. The minipackets in a queue are linked to each other in the order of arrival by writing the pointer of the new minipacket in the link buffer memory **32** of the last minipacket in the link-list queue. To increase the flexibility of access to the linked-list, a separate link buffer memory **32** is used. Both the minipacket memory and the link buffer memory are accessed using the same pointer.

The queue control part **10** in Figure 2 can include a queue information controller **26** that maintains per-queue control information, such as empty/nonempty state, packet arrival time, queue length, average queue length, and the last time a queue was serviced. The queue information controller uses the queue information memory **34** in Figure 3 to store and retrieve information. The collection and use of some of this control information is optional and could vary in different implementations of the present invention. The queue control part **10** in Figure 2 can also include a buffer management controller **28** that can

process information generated by the queue information controller to determine whether a data packet should be discarded or marked for certain treatment. Alternatively, the information in the queue information controller 26 can be reported to an attached processor, that uses this information for switch management purposes.

The sequence of functions performed upon arrival of a minipacket into the queue control part of the present invention according to a typical example implementation of the queue control part is as follows and is shown in Figure 4. The minipacket and associated information is stored in a free block of memory in the minipacket memory 30. The pointer for the location of the minipacket is linked to the tail of the linked-list queue whose index is attached to the minipacket, and the linked list 32 is updated accordingly. The queue information 34 for the given queue index is updated as well. This information includes a time stamp that is assigned to the minipacket according to a global clock that indicates the current time of the system. The length of the queue is incremented and stored back. Optionally, a running average of the queue length can be updated. The age of the queue, which is defined as the time stamp of the head-of-line minipacket in the queue, is updated if the minipacket arrives to an empty queue.

The sequence of functions in the example implementation of the present invention to transfer a minicell to the head-of-line sequencer part 14 in Figure 4 is as follows. The HoL selector 12 obtains the queue index of the queue that is selected. The queue index is used to access the head pointer buffer memory 31. The head pointer is used to read out the first minipacket in the queue and the linked–list is updated accordingly. A minicell is prepared containing the subset of the information in the minipacket that is required by the HoL sequencer part 14. Said information contains the pointer to the minipacket location in the minipacket memory 30. Said minicell is transferred to the HoL sequencer part. In this particular implementation, the minipacket is kept in the minipacket memory 30 even though it is removed from the link buffer memory 32. The queue length is updated by reading the queue length, decrementing it and writing it back into the entry for the given queue index in the queue information memory 34. The age of the queue is also updated by writing the age of the new HoL minipacket in the queue age entry for the

given queue index in the queue information memory. The age of the new HoL minipacket is obtained from the information stored for the HoL minipacket in the minipacket memory **30**.

The series of functions to output a scheduled minipacket in the queue control part **10** of the example implementation of the present invention is as follows. We refer to the minicell that is selected by the HoL sequencer part **14** for transmission as the winner minicell. The winner minicell is passed to the queue control part **10** by the HoL sequencer. The pointer in the minicell is used to read the actual minipacket out from the minipacket memory **30**. The minipacket is then sent out through the output interface **18** of the scheduler system. The pointer of the minipacket is returned to the pointer pool of free memory that can be used to store new minipackets and that is maintained by the memory space controller **21**. As an option, it is possible to send out from the scheduler system only those parts of the minipacket that are required by the switch or router system. As an option, the exiting minipacket may contain the timestamp of instant of arrival to the scheduler system and the instant of departure from the scheduler system, or the difference thereof.

The functions in the queue control part, system memory part, and HoL selector part of the present invention are performed by separate units and in parallel wherever it is possible, and if necessary serially within a series of time slots (clock cycles), such as in cases where two or more functions access the same memory or register.

An optional buffer management controller **28** of the queue control part **10** can use the per-queue contained in the queue information memory **34** to discard minipackets from the queues if necessary, or to mark minipackets for certain subsequent treatment in the scheduling system. Different discarding algorithms can be implemented in said buffer management section, for example the minipacket at the head or the tail of a queue can be discarded. Different criteria can be used to decide on when to discard a minipacket, e.g. queue size and pre-established thresholds. Packet discarding as in RED and RIO can also be implemented by comparing a pseudo-random number and a threshold that depends on

the average queue size [Floyd 1993], [Clark 1998]. The buffer management controller may also be used to police arriving packet flows. The queue information memory and associated processing is used to police packet arrival patterns. For example, the Generic Cell Rate Algorithm for ATM cell arrivals can be policed in this fashion [DePrycker pg 305]. This discarding procedure is performed upon arrival of a minipacket for a queue. The discarded minipackets are marked in a unique manner and output by the scheduler system. The original data packet is then discarded by the switch, router, or subsystem thereof.

In this invention a HoL selector part **12** transfers minicells representing the head-of-line minipackets to the HoL sequencer part **14** of the system. Each subclass of packets that belong to a given queue is defined so that only first-in first-out scheduling is required for the minipackets within the queue. Consequently, each time the HoL sequencer needs to select the next minipacket to be transferred out of the entire scheduler system, only the head-of-line minipacket of each queue needs to be considered. The function of the HoL selector part **12** is to ensure that the HoL sequencer part **14** has the appropriate head-of-line minicell from each non-empty queue. Each time a winner or discarded minicell is output by the HoL sequencer, the HoL selector part transfers to the HoL sequencer the next HoL minicell of the queue with the same queue index. As a result of the implementation of this mechanism the size of the required space in the HoL sequencer part is equal to the number of queues (distinct indices) in use, rather than the total number of minipackets in the system. Typically the total number of minipackets in the system is much larger than the number of queues.

Table II below is a pseudocode representation of a HoL Selector and sequencer:

```
HOL Selector and HOL Sequencer:


If      (there is a minicell departure from the sequencer)        then
```

```
Send the minicell to Queue Control
Get the index of the departing minicell

If      (queue of the index is not empty)    then

        Dequeue the HOL minicell
        Tag the HOL minicell
        Enter the tagged minicell into the sequencer circuit

        If      (queue becomes empty)       then

                Mark the queue as empty

        Endif

 Else
        Mark the queue as not represented in the HOL Sequencer

 Endif

Endif

For     k

 If      (there is a new arrival from Buffer Management part)          then

        Tag the new arrival minicell
        Enter the new arrival minicell into the sequencer circuit

 Endif
```

Endfor

The HoL selector mechanism is an important aspect of the present invention. As shown in Figure 5, the HoL selector part 12 of the present invention contains a HoL controller 40 that ensures that the new HoL minipacket of the most recently serviced queue is sent to the HoL sequencer part in time for the next scheduling decision. The following describes one implementation of the mechanism. However it must be noted that this implementation is not the only way to realize the mechanism. The basic goal of this mechanism can be achieved through other implementations.

Each of the queues has a flag bit associated with it that indicates whether or not the queue is represented in the HoL sequencer part. For example, this flag information can be stored in the queue information memory 34. An arriving minipacket is placed in the queue if the flag is already set to 1. A minipacket arriving to a queue with the flag bit set to 0 always results in the sending of a corresponding minicell to the HoL sequencer part, and the flag is set to 1. The direct transfer of said minicell to the sequencer part is essential to maintain the correct operation of the scheduling algorithm. A possible implementation of the direct transfer mechanism involves setting aside time slots so that each arriving minipacket can have a corresponding minicell transferred directly to the sequencer part if necessary. However it must be noted that this implementation is not the only way to realize the mechanism.

Whenever a minipacket is scheduled for service in the HoL sequencer part of the present invention, the minicell representing the minipacket is sent back to the HoL selector part where the HoL controller 40 extracts the queue index in the minicell and sends the next HoL minicell of the same queue to the HoL sequencer part before the next scheduling decision time for the related output port. If the queue is empty, the HoL controller sets the

flag bit to 0 to indicate that the queue is not represented in the HoL scheduler part. This mechanism is completed by the procedure for empty queues that is explained next.

The winner minicell is also sent to the queue controller part 10, where the corresponding minipacket is removed from the minipacket memory and then output back to the switch or router or subsystem thereofr.

An important aspect of the present invention is the way empty queues are handled. A major limitation in queuing systems that incorporate a large number of queues is the need for updating the state of the queuing system to track the state transitions from an empty queue to a non-empty queue or vice versa. A major problem with these systems is that whenever the queue that is chosen for service is empty, many time slots are required before another non-empty queue can be scheduled for service. In the present invention there is no need to examine the state of all the queues at every cycle of operation. A queue is examined only at the time that a new minipacket arrives for that queue or when a minicell from that queue is serviced. This aspect of the present invention can also be implemented in different ways. Here we describe one implementation of the mechanism to elaborate on fundamental objectives of the mechanism.

An empty queue flag bit associated with each queue indicates whether or not the queue is empty. The empty queue flag can be stored in the entry for a given queue index in the queue information memory 34. In this mechanism whenever a new minipacket arrives to an empty queue that is already represented, the minipacket is put in the linked-list queue and the empty flag is set to 1. If a minipacket arrives to an empty queue that is not represented in the sequencer part it is sent to the HoL sequencer part. In this case the empty flag bit remains 0, however the represent flag bit is set to 1. Whenever a minicell is scheduled for service, the next HoL minicell of the serviced queue is sent to the HoL sequencer part, and if the minicell is the last in the queue, the empty queue flag is reset to 0 to indicate an empty queue. In the case if the queue is already empty the flag bit

remains 0, however the represented flag bit is set to 0 to indicate the queue is not represented in the HoL sequencer part.

The HoL selector part of the present invention can also provide shaping and rate regulation using a control circuit that marks minicells for certain treatment in the HoL sequencer part ,or that enables or disables the transfer of certain HoL minicells to the HoL sequencer part. The control circuit can be implemented in different ways. For example per-queue token bucket regulators can be used to determine the time at which an HoL minicell may be transferred to the HoL sequencer part [Ferguson 1998]. The control mechanism is incorporated in the present invention by a simple modification that can be implemented into the HoL controller: The next HoL minicell is transferred only if sufficient tokens are available for the given packet length. Otherwise the normal replacement of the HoL minicells is not performed. The policing and shaping circuit records the indices of the queues whose HoL replacement has not been performed because of the lack of the tokens. The HoL replacement of a minicell is resumed when sufficient tokens become available for the transfer of the HoL minicell. The operation of the shaping and policing circuit operates in parallel, and independently of the HoL controller so that speed of operation is unaffected. The implementation of this control circuit is optional. Furthermore, rate control can also be implemented directly in the sequencer part by using rate-based scheduling algorithms [Hashemi 1997b]. Alternatively said token bucket regulators can set an earliest service time field in the minicell that indicates to the HoL sequencer part the time when a minicell is eligible to exit the system.

The HoL sequencer part 14 of the present invention consists of a tagging circuit and a sequencer circuit as shown in Figure 6. A tagging circuit takes a HoL minicell and associated information from the HoL selector part and produces a tag. The sequencer circuit sorts the HoL minicells according to a specific scheduling algorithm and at the appropriate time instants outputs a winner minicell which is then transferred to the HoL selector part and to the queue control part.

The HoL sequencer can handle the scheduling of variable length packets as follows. Since an output port will not become available for the next transmission until an entire packet is transmitted, the next minicell is not allowed to exit the system until said transmission time is completed.

The sequencer of the present invention orders the minicells based on their tags. Therefore a scheduling algorithm can be implemented by tagging minicells in such manner that the minicells are sorted in the same order that they would be chosen by the specific scheduling algorithm. The tagging algorithm in the tagging circuit and the ordering algorithm in the sequencer need to be selected jointly so that the desired scheduling algorithm can be realized. The tagging circuit can be designed and implemented in different ways and can be customized for specific scheduling algorithms.

The tagging circuit can be very simple. In the case were minicells carry static priority values, the circuit only adds a number of flag bits required by the sequencer circuit to perform the basic sorting operation. On the other hand for sophisticated scheduling algorithms, complex processing may be required in the tagging section that can be provided by arithmetic circuits in hardware or by local processors or through an attached processor. For example, weighted fair queuing involves the calculation of a tag that depends on the packet length, the weight of the queue, and a virtual time of the given queue. The information required to calculate the tag may be carried by the minicells or transferred in parallel to the HoL sequencer part. It is also possible to store per queue information required for tagging locally in the tagging section. An interface to an attached processor can be used to initialize or dynamically update this information.

Any sequencer that can select the minicell with the highest tag value can be used in the HoL sequencer part. An important aspect of the current invention is the use of a sequencer circuit that maintains a sorted list of minicells according to tag values. Various sequencers that maintain said sorted list of minicells have been disclosed in [Hashemi 1997a], [Hashemi 1997b], [Hashemi 1997c], [Hashemi 1997d] and [Leon-Garcia and Hashemi, "The Single Queue Switch," Canadian Application No. 2,227,655].

These disclosures have demonstrated that a broad range of scheduling algorithms can be implemented by using said sequencer circuits that sort minicells according to numerical and logical comparison of binary tags. Said sequencer circuits have the advantage that the next HoL minicell can be determined by a simple comparison of the new incoming minicell and the minicell that is currently at the head of the line in the sequencer. The new winner minicell is determined in the time required to make this single comparison. This minimal comparison time allows for very high speed scheduler implementations of the current invention.

We describe a novel generalized sequencer circuit **52** in Figure 7 to be used as the sequencer circuit **51** of the present invention. The generalized sequencer circuit of the present invention uses the principles of the sequencer circuit described in [Hashemi 1997a]. The generalized sequencer of the present invention consists of a chain of buffering units **53** in Figure 7. Each buffering unit can accommodate two minicells. Each minicell, as described before, is a fixed-length block of data that includes a tag. Minicells residing in the chained buffering units form a sorted list. Minicells in the sorted list can travel from one unit to the adjacent units in forward and backward directions.

Each buffering unit **53** has a comparison unit **55** that can compare the two tags of the two minicells residing in the buffering unit. Comparison can involved an arithmetic or logical operation. Moreover, as an important aspect of the present invention, each buffering unit has an arithmetic and logical unit **56** that can perform arithmetic and logical operations on the tags. At each cell-time a new minicell enters the queue from the head of the queue. The cell is compared to the cell at the head of the queue. According to the predefined logic of comparison and the two tag values, one of the two minicells is sent out as the winner. The other minicell, the loser, is sent one step back inside the queue to the next buffering unit. According to the present invention an arithmetic and/or logical operation can be performed on the tag of the minicell that is sent back in the queue.

The comparison and forwarding procedure is repeated at every buffering unit spreading into the buffer, like a wave, until the last unit is reached. In this manner, the winner of

the unit i is always forwarded to the unit i-1 (the forward path), and the loser to the unit i+1 (the backward path) as is shown in Figure 7. The next new minicell enters the sequencer circuit immediately after the previous minicell has departed the first buffering unit, thus generating a new wave. The events inside the sequencer are exactly synchronized. Therefore, whenever the new wave arrives at unit i, the outcome of the previous wave is already in unit i+1 and the comparison can start immediately. Each arriving minicell moves backward into the queue until it finds its right place in the queue pushing other minicells backward into the queue.

A blocking capability **57** in Figure 7 is implemented in the sequencer, so that minicells can enter the sequencer even when a minicell is not allowed to exit the sequencer. To realize this capability, the winner in the comparison in each unit remains at the same unit instead of being forwarded to the adjacent unit. The loser is sent backward as before. In the scheduling of variable length packets, the blocking capability is used to determine the timing of the exit of the next winner minicell.

An important feature of the present invention is that the generalized sequencer circuit can use the arithmetic and logical units **56** in Figure 7 to recalculate minicell tag values "on the fly" as the minicells are flowing inside the sequencer. The feature is required in scheduling algorithms where the arrival of a packet to a queue can trigger the recalculation of tags in related minicells. As an example, in scheduling algorithms such as hierarchical weighted fair queuing [Zhang 1995] recalculations may be required upon arrival of a minicell to an empty queue. In this case the insertion of a minicell with appropriate flag information indicates an arrival to an empty queue can trigger the recalculation of tag values of related minicells as the wave generated by the arriving minicell propagates through the sequencer.

An important feature of the novel generalized sequencer circuit is the capability to maintain several logical subqueues within the same physical sequencer circuit. For example as shown in Figure 8, each logical subqueue can represent minicells belonging

to a given priority class. Additional functions can be implemented within each logical subqueue such as age control, discarding over-aged cells and so on.

The basic mechanism in the generalized sequencer is flexible enough to perform different scheduling schemes by simply changing the tagging algorithm. As an example, weighted fair queuing and priority-based algorithms can be handled by the same hardware. In weighted fair queuing, the tag reflects the finish number of a data packet, which is determined upon its arrival, and the minicells are served in the order of their finish numbers [Keshav page 239]. In the second case, the tag reflects the priority level of a cell and the packets are served in the order of their priority levels [Hashemi 1997a]. In either case the comparison hardware determines the larger of two numerical values.

The tag can be composed of several distinct fields, each controlling an aspect in the sequencing of the cells. For example, an age field can be used, in conjunction with the priority field, to arrange cells of the same priority in order of age as shown in Figure 8.

In addition, programmable options can be implemented in the hardware and controlled by flag bits in the tag. For example, discarding over-aged cells, aging, and joining of the priority and age fields together can be enabled or disabled by using flag bits in the tag field of each cell.

As a further step, regarding the advances in the technology of Field Programmable Gate Arrays [**Brown 1992**] and other field-programmable logic devices, programmable logic can be used for the comparison logic units in the sequencer, allowing flexibility in the implementation of scheduling algorithms. In particular, a sequencer circuit with generic comparison logic can be manufactured and later customized for specific scheduler implementations by appropriately programming the logic section. For example, the exact definition of the tag field, which includes the type, length, and position of each field in the tag relating to the properties such as the age and priority can be defined according to the desired scheduling algorithm after the manufacture of the circuit. In addition, the

operations used to compare and operate on the tags can also be defined after the manufacture of the circuit.

An important aspect of the present invention is the capability of scheduling of data packets that are destined to different output ports using a single scheduler system. A generalized single-queue sequencer circuit is used as the scheduler circuit of the HoL sequencer part of the present invention to realize the aspect of the present invention. A single-queue sequencer circuit is described in [Leon-Garcia and Hashemi, "The Single Queue Switch", Canadian Application No. 2,227,655]. A generalized single-queue circuit is obtained by replacing the buffering units of the single queue sequencer circuit with the novel buffering units **53** of the generalized sequencer circuit **52** of this invention. The generalized sequencer circuit **52** can schedule the transfer of data packets from a multiplicity of input ports to a single output port. The generalized single-queue sequencer of this invention can combine N generalized sequencers, each dedicated to a unique output port, into a single sequencer circuit.

In the case where the scheduler system handles data packets destined for a multiplicity of output ports, the sequencer circuit in the HoL sequencer part of the present invention has a multiplexer **61** and output controller 62 that manages the input and output of the generalized single-queue sequencer as shown in Figure 9.

The generalized single-queue sequencer **60** in Figure 9 uses the principles of the single-queue switch architecture described in [Leon-Garcia and Hashemi, "The Single Queue Switch", Canadian Application No. 2,227,655] and Hashemi [1997d] and introduces a novel buffering element that includes arithmetic and logical processing. In the single-queue switch the N output queues that correspond to N output ports are interleaved into a single sequencer circuit using a grouping algorithm.

The generalized sequencer **52** of Figure 7 has the capability of organizing the logical queues within the same physical queue. In the generalized single-queue sequencer **60** of Figure 9 the same capability is used to interleave the queues of different output ports into

the same physical sequencer circuit. The minicells destined for a given output port are said to belong to the same logical queue. The interleaving mechanism is as follows. The sequence of minicells is divided into groups. The first group contains the first minicell of each logical (output) queue. The second group contains the second minicell of each logical (output) queue and so on. Within each group the cells are placed in order of increasing logical output queue number. An example of the interleaved sequence of the minicells is shown in Figure 10. If the logical queue of one of the output ports has only k cells, no place is reserved for that output in groups k+1 and up. In this manner, in each group only output ports that have a minicell for that group will occupy a place. However, in the above example, if a new minicell arrives for the aforementioned output port, it will be inserted in the appropriate place in the group k+1, pushing the rest of the minicells one step back in the queue. As a result, in this circuit all of the buffering spaces are shared by all of the output ports.

The grouping mechanism is implemented simply by exploiting the basic function of tagging and tag comparison in the generalized sequencer. Every minicell entering the generalized single-queue sequencer carries a field in its tag that indicates its output port number. Inside the generalized single-queue sequencer the output port number field of the arriving minicell is compared to the existing minicells in each group. If there is a minicell with the same port number in the group, the minicell is sent to the next group. Within a group, a minicell is inserted in order of increasing output port number. This ordering is accomplished by comparison of the output port number field of a new minicell and minicells in a group. A method for detecting the groups, and a method for sending a minicell from one group to the next group using two flag bits in the tag and an associated grouping algorithm has been described in [Leon-Garcia and Hashemi, "The Single Queue Switch", Canadian Application No. 2,227,655]. and [Hashemi 1997c].

The generalized single-queue sequencer operates according to scheduling rounds of fixed duration. In an M input by N output switch or router or subsystem thereof, up to M minicells can enter a scheduler system during a scheduling round. On the output side, one group of minicells leaves the single-queue sequencer every scheduling round. The

minicells exit the sequencer sequentially in order of output port number. The scheduler system outputs a maximum of N minicells in a scheduling round when the HoL group has one minicell for each output port. When the HoL group does not contain N minicells, then one or more of the output ports will not be utilized during a scheduling round. The time to transfer the group remains constant. The output will remain idle during the turns of the absent minicells. Obviously, the queue will not move forward during this time period necessitating the use of the aforementioned blocking mechanism **63** in Figure 9.

The generalized single queue sequencer can handle variable length packets as follows. If the port corresponding to a given minicell in the HoL group has not completed its previous packet transmission, then said minicell is sent back into the generalized single queue sequencer.

Despite the fact that a multiplicity of logical queues are interleaved in a single physical generalized single-queue sequencer circuit, each logical queue can be viewed as an independent virtual queue operating on a distinct generalized sequencer. Each logical queue can then implement a different scheduling algorithm by invoking different features of the comparison logic and arithmetic logic in each buffering element. The tagging circuit must operate on each minicell according to the scheduling algorithm that corresponds to the specific minicell. Each minicell must carry flag bits that invoke the appropriate processing in each unit.

We can summarize the procedure that transpires as a "transit" minicell enters a buffering unit in the generalized single-queue sequencer. The output port field of the transit minicell is compared to the output port field of the minicell in the buffering unit. If the two output port fields are different, the transit field is declared a loser and is passed to the next unit. If the two output port fields are the same, the priority fields of the two minicells are compared to each other first. If the transit minicell is the loser, the procedure continues as before. If the transit minicell is the winner, then the transit minicell is placed in the unit and the resident minicell is pushed out. In this case the loser minicell is passed to the next group.

A multicasting mechanism can be incorporated in the multiple-port implementation of the present invention. According to the multicasting mechanism of the present invention, a packet can be scheduled for transmission to its destination output ports independently, while keeping only one copy of the original data packet outside the scheduler system and using only one minicell in the scheduler system. The multicasting capability of the scheduler of the present invention can be used for multicasting in ATM switches. It can also be exploited as a complementary facility for multicasting in IP routers. The details of the multicasting mechanism is as follows.

Multicast minipackets are put in a different set of queues than unicast minipackets by the queue control part. The HoL minicell of each multicast queue is sent to the sequencer circuit. These multicast minicells are treated by the generalized single-queue sequencer circuit as belonging to a separate virtual queue. In particular, the multicast logical queue is treated as if it corresponds to the queue for a fictitious output port with a number (say port #0) that precedes the number of all other output ports.

The first timeslot of every scheduling round of the generalized single-queue sequencer is dedicated to the virtual multicast port (port #0). The head of line multicast minicell in the first group of the sequencer circuit is sent to a multicast controller module 72 of Figure 11. The destination list of said multicast minicell is retrieved and is stored in a register in the multicast controller as a bitmapped list. The destination list can be part of the minicell set by the routing protocol in the switch. The list can also be retrieved from a local look up table set per connection for connection-oriented switches such as ATM.

During the timeslot that is dedicated to each output port, the HoL unicast minicell for said output port is sent to the multicast controller. In the multicast controller if the output port is one of the destinations of the multicast minicell, a copy of the multicast minicell is sent to the output by the output controller 71 and the related bit is reset to 0. The HoL unicast minicell is sent back to the sequencer in this case using multiplexer 74. If the destination

list becomes empty in a given time slot, the copy of the multicast minicell that is sent out is marked as the final copy of the multicast minicell.

Whenever a copy of a multicast minicell is sent out by the HoL sequencer, the minipacket corresponding to said minicell is output from the scheduler system. The corresponding minipacket is not removed from the minipacket memory if the minicell is not the final copy of a multicast minicell. If the minicell is the final copy of a multicast minicell, the HoL selector transfers the next HoL minicell of the same multicast queue to the HoL sequencer part. The queue control part clears the minicell from the memory by recycling the pointers similar to pointers of normal minicells. The minipacket that is output from the scheduler system is marked to indicate that it corresponds to the last copy of the data packet.

Packets can be discarded in the HoL sequencer part of the present invention. According to the discarding mechanism of the present invention a minicell can be marked as discarded. A discarded minicell is then output from the scheduler system so that the original data packet can be discarded. The discarding mechanism is implemented as follows.

In the generalized sequencer as well as in the generalized single-queue sequencer, it is possible to mark a minicell as discarded by setting a flag bit in the tag. For example, a minicell may be marked as discarded if it is found to exceed a preset age limit. Such a minicell exits from the back of the sequencer circuit. A discard control circuit sends such a minicell back to the queue control part, where the corresponding minipacket is removed from memory, and is then output from the scheduler system. The corresponding packet can then be discarded. The HoL minicell replacement is performed for discarded minicells in the same manner as for normal minicells. A discarded multicast minicell is handled in the same manner as a discarded unicast minicell.

The present invention can be integrated in a switch or router or subsystem thereof to queue and schedule the transmission of packets on one or a multiplicity of output ports. The integration of the system of the present invention in switching and routing systems can be done in different ways depending on the architecture of the switching or the routing system.

First we consider a centralized queue controller and scheduler for a multiport switch. Figure 12 shows a block diagram of a switching system **80** that incorporates the present invention as a centralized queuing and scheduling controller. Data packets arriving from a multiplicity of input ports **82** are stored and for each packet a minicell is sent to the controller. The minicell contains information regarding the class, priority, output port number and a pointer to the storage location of the data packet. Such a minicell can be composed and represented to the queue controller and scheduler module in different ways. However the location of each field of the information must be known to the interface logic that is used to integrate the module in the system.

The minicells are queued and scheduled based on the information contained in them. The scheduler system selects a minicell for each output port during each scheduling period. Each scheduling period contains one time-slot per output port. Selected "winner" minicells are transferred to the switch fabric during their corresponding time-slots. The minicell refers to a packet that is stored in the fabric. The packet is then retrieved and is transmitted to the output port according to the switch fabric architecture. The speed of the queuing and scheduling controller must be sufficiently fast to handle the N ports in this manner.

In an alternative implementation, a classifier circuit can be used to perform packet classification and to provide class and priority information required by the queuing and scheduling controller module. The classifier receives the packets or the headers of the packets or the information in the headers, and determines the priority, class, or type of the packet, and along with the destination port number sends the information and a queue index to the queuing and scheduling system.

In an alternative implementation the classifier circuit can be combined with the queuing and scheduling controller circuit.

The multiple-port queuing and scheduling system can also be used in a multiport line card in which the queuing and scheduling system handles the buffer management and scheduling of data packets destined for a multiplicity of output ports on the line card. As shown in Figure 13 data packets are transferred from the switch fabric of a switch of router to an output line card **80**. The data packets are stored in memory and the scheduler system **81** is used to implement buffer management, scheduling, and other packet management functions. The scheduler system determines the order in which data packets are transmitted in the multiplicity of the output ports **82** in the line card. A switching or router may have several line cards each containing a number of output ports.

The queuing and scheduling system of the present invention can be used in a single port line card **85** in Figure 14for queuing and scheduling the packets destined for an output port on a line card. A generalized sequencer circuit is can be used as the sequencer circuit of the scheduling part of the present invention in this case.

The queuing and scheduling controller of the present invention as described in the above example can be implemented as a single-board card that can be added to a system such as a personal computer that is used as a switch or router. The queuing and scheduling controller directs the transfer of data packets from the input line card across a standard bus such as PCI (Peripheral Component Interconnect) to the output line card.

The single-port and multi-port queuing and scheduling system 89 can also be used in an input line card 88 to process packets as they enter a switch or router and prior to transfer across a switching fabric as shown in Figure 15. The arriving data packets are stored in memory and the queuing and scheduling system implements policing, metering and buffer management functions. The queuing and scheduling system may also be used to schedule the order in which packets are transferred across the switch fabric. The multi-

port queuing and scheduling system can be used to schedule the transfer of packets in switch and router designs that use several parallel fabrics to transfer packets between line cards.

Although the invention has been described with reference to certain specific embodiments, various modifications thereof will be apparent to those skilled in the art without departing from the spirit and scope of the invention as outlined in the claims appended hereto.

1. References

[Floyd 1993] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance, *IEEE/ACM Transaction on Networking,* August 1993.

[Keshav] S. Keshav, *An Engineering Approach to Computer Networking,* Addison-Wesley, 1997.

[Peterson] L. Peterson and B. Davie, Computer Networks, Morgan Kaufman, 1996.

[Zhang 1995] H. Zhang, "Service Disciplines for Guaranteed Performance Service in Packet Switching Networks," *Proceedings of the IEEE,* October 1995

[Floyd 1995] S. Floyd and V. Jacobson, "Link-sharing and Resource Management Models for Packet Networks," *IEEE/ACM Transaction on Networking,* August 1995.

[Bennett 1996] J. Bennett and H. Zhang, "Hierarchical Packet Fair Queuing Algorithms," IEEE/ACM Transactions on Networking, 5(5):675-689, Oct 1997. Also in Proceedings of SIGCOMM'96, Aug, 1996

[Kumar] V. Kumar, T. Lakshman, D. Stiliadis, "Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow's Internet, *IEEE Communications Magazine, May 1998.*

[Newman] P. Newman, G. Minshall, T. Lyon, L. Huston, "IP Switching and Gigabit Routers," *IEEE Communications Magazine, January 1997.*

[Keshav 1998] S. Keshav and R. Sharma, "Issues and Trends in Router Design," *IEEE Communications Magazine, May 1998.*

[Hluchyj] "Methods for prioritizing, selectively discarding, and multiplexing differing traffic types fast packets," US Patent 5231633]

[Kai Eng] "Controller for input-queued packet switch," US Patent 5255265].

[Hashemi 1997a] M. Hashemi and A. Leon-Garcia, "A General Purpose Cell Sequencer/Scheduler for ATM Switches," *Inforcom '97*, Kobe Japan, April 1997.

[Hashemi 1997b] M. Hashemi and A. Leon-Garcia, "Implementation of Scheduling Schemes using a Sequencer Circuit," *Proc. Voice, Video and Data Communications, SPIE*, Dallas, November 1997.

[Hashemi 1997c] M. Hashemi and A. Leon-Garcia, "The Single Queue Switch," *Inforcom '97*, Kobe Japan, April 1997.

[Leon-Garcia and Hashemi, "The Single Queue Switch", Canadian Application No. 2,227,655].

[Hashemi 1997d] M. Hashemi and A. Leon-Garcia, "A RAM-based Generic Packet Switch with Scheduling Capability," *Proc. IEEE Broadband Switching Symposium '97*, Taipei, December 1997.

[Zhang 1999] L Zhang, Brent Beacham, M. Hashemi, Paul Chow, and A. Leon-Garcia, "Design and Implementation of a Scheduler Engine for a Programmable Packet Switch," *Hot Interconnects 7*, Stanford University, August 1997.

[Clark 1998] D. Clark and W. Fang, "Explicit Allocation of Best Effort Packet Delivery Service," *IEEE/ACM Transactions on Networking, August 1998.*

[DePrycker] M. DePrycker, *Asynchronous Transfer Mode*, Prentice Hall, 1995.

[Ferguson 1998] P. Ferguson and G. Huston, *Quality of Service*, Wiley, 1998.

[Brown 1992] S.D. Brown, R.J. Francis, J. Rose, Z.G. Vranesic, "Field-Programmable Gate Arrays", , Kluwer Academic Publishers, 1992

[Bonomi et al, "Method for Integrated Traffic Shaping in a Packet-Switched Network," U. S. Patent Number 5,864,540]

[Rexford et al, "Scalable Architectures for Integrated Traffic Shaping and Link Scheduling in High-Speed ATM Switches," *IEEE Journal on Selected Areas in Communications*, June, 1997, pp938-950.

THE EMBODIMENTS OF THE INVENTION IN WHICH AN EXCLUSIVE PROPERTY OR PRIVILEGE IS CLAIMED ARE DEFINED AS FOLLOWS:

1. A method for scheduling routing of packets between an input port and an output port, wherein each packet has an index that specifies both a unique destination output port for the packet and membership in a subclass, the method comprising the steps of:
   (a) creating for each arriving packet, a minipacket including the index of the packet, a unique identifier, and scheduling information;
   (b) assigning to each index a unique queue;
   (c) queuing said minipackets in order of arrival in its corresponding unique queue determined by the index;
   (d) selecting the next minipacket to be transmitted from among the set of all head of line minipackets according to a given scheduling algorithm; and
   (e) transmitting the packet corresponding to the selected minipacket.

2. A method as defined in claim 1, said unique identifier is the packet's storage location.

3. A method as defined in claim 1, said queuing being performed by a queue control module that queues and stores said minipackets in order of arrival in a unique queue that is assigned to each index.

4. A method as defined in claim 1, said queues are implemented using a bank of linked-list queues that can be assigned flexibly and arbitrarily to the minipackets in the system.

5. A method as defined in claim 4, said bank of linked-list queues is scalable.

6. A method as defined in claim 1, said queue control module for implementing a plurality of buffer management and policing mechanisms.

7. A scheduler for scheduling routing of packets between an input port and an output port, wherein each packet has an index that specifies both a unique destination output port for the packet and membership in a subclass, the scheduler comprising:
   (a) a memory for storing minipacket information corresponding to an arriving packet, including the index of the packet, a unique identifier, and scheduling information;
   (b) a queue control for assigning to each index a unique queue and for queuing said minipackets in order of arrival in its corresponding unique queue determined by the index;
   (c) a selector for selecting minipackets for transmission from each queue according to a predetermined selection criteria; and
   (d) a sequencer for selecting the next minipacket to be transmitted from among the selected minipackets according to a predetermined scheduling algorithm.

8. A scheduler as defined in claim 7, said queues being FIFO queues.

9. A scheduler as defined in claim 8, said minipackets being selected from the head of said queues.

10. A scheduler as defined in claim 8, said queues being linked lists.

11. A scheduler as defined in claim 8, said sequencer transmitting the packet corresponding to one of a group of minipacket selected from the head of line for each queue.

12. A scheduler as defined in claim 8, including a buffer manager.

13. A scheduler as defined in claim 8, said sequencer including a policing module.

14. A scheduler as defined in claim 8, said sequencer including a shaping function.

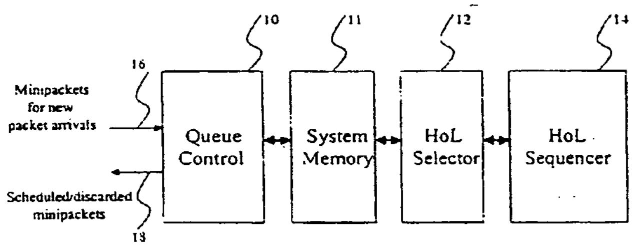


Figure 1 Block diagram of a scheduler system

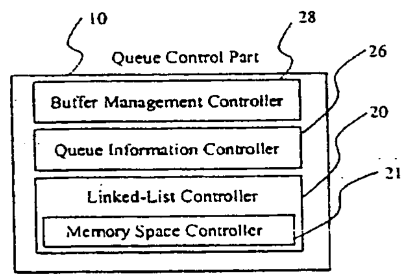


Figure 2 Queue controller

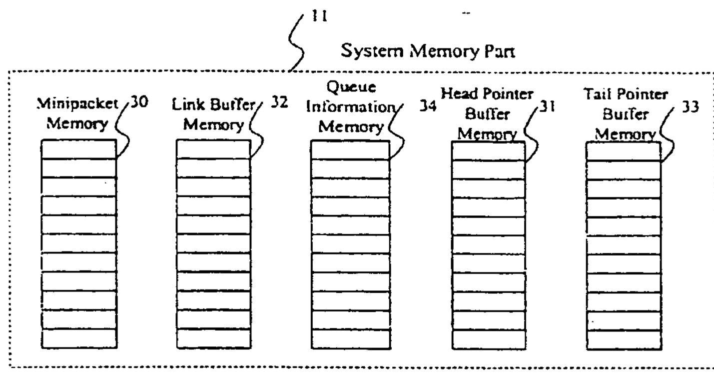


Figure 3 System Memory

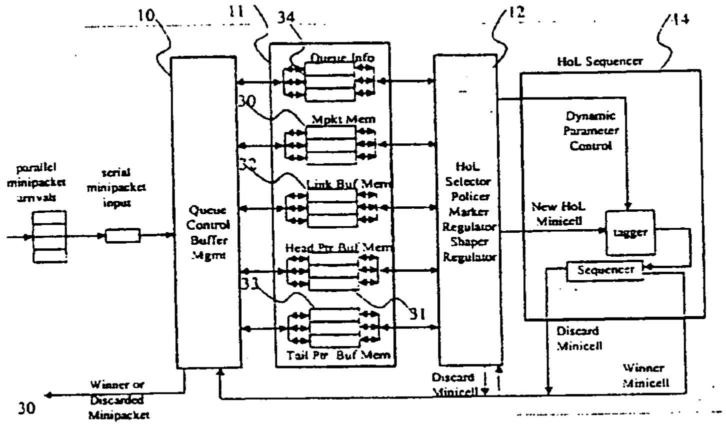


Figure 4 Example Implementation of Packet Scheduler System

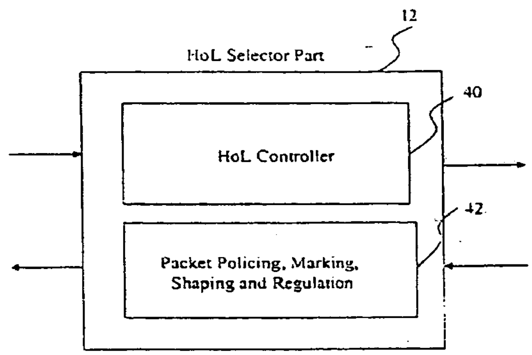


Figure 5 HoL Selector

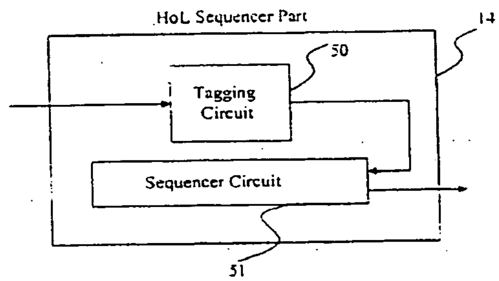


Figure 6 HoL Sequencer

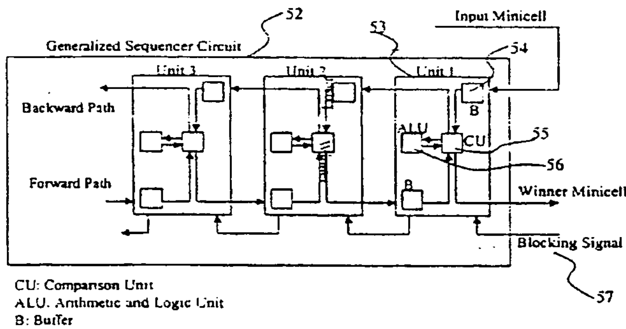


Figure 7 Generalized Sequencer Circuit

CA 02290265 1999-11-24

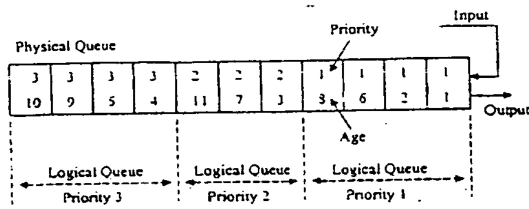


Figure 8 Subqueues with Sequencer Circuit

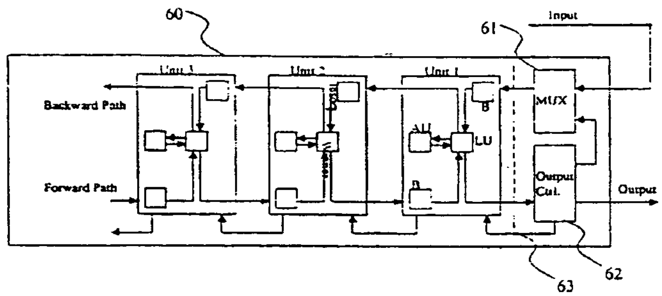


Figure 9 Generalized Single-Queue Sequencer Circuit

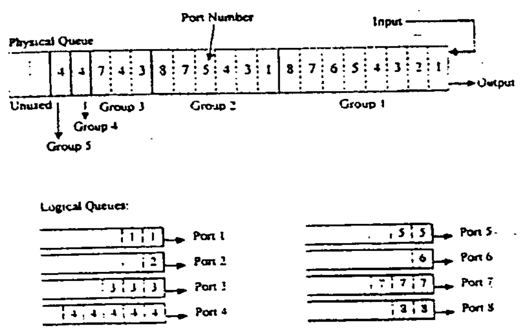


Figure 10 Example interleaving the logical output queues for 8 ports

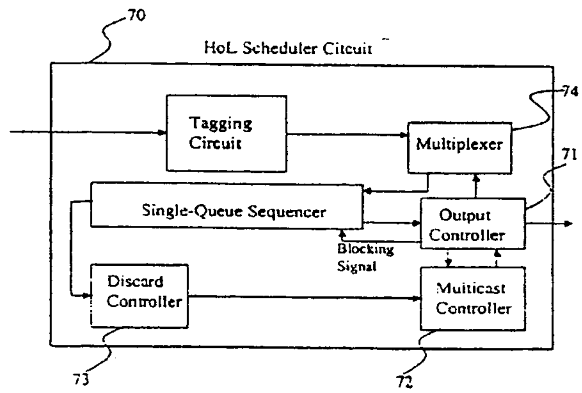


Figure 11 HoL Sequencer with Multicast Controller

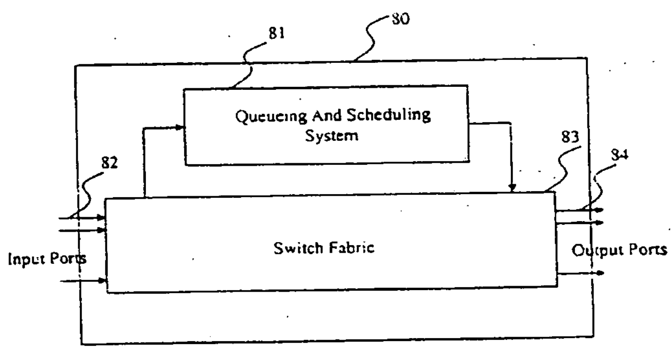


Figure 12 Example application of Scheduling System in Centralized Switch

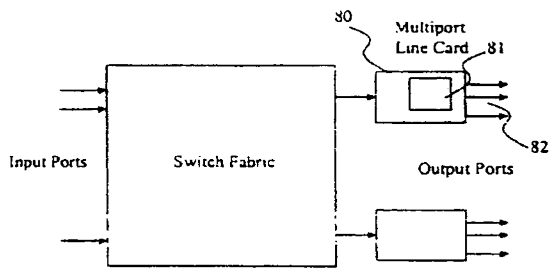


Figure 13 Example application of Scheduler system in Multiport Line Card

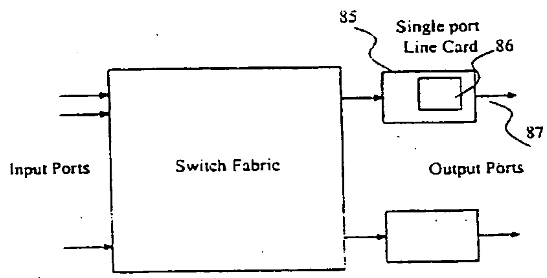


Figure 14 Example application of scheduler system in Single port line card

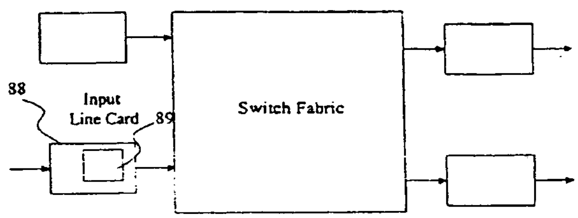


Figure 15. Queue and Scheduling system in Input Line Card